## Case1 v1:

Observations: Lidar scan (1080) and previous action(1).

Actions: accelerate, brake, cruise.

Rewards: 0 for every step. -100 for not completing. -50 for crash. 600 – time taken for completion of a lap

Terminated: when 2 laps are completed

Truncated: when car crashes.

Notes about the implementation: learning was done while the car was running.

NN: self.fc1 = nn.Linear(1081, 512)

self.fc2 = nn.Linear(512, 128)

self.fc3 = nn.Linear(128, 64)

self.fc4 = nn.Linear(64, 3)

Notes for modification: The Model was not learning to brake before a turn.   
Lap times were not competitive.  
Previous actions should not matter for current actions.

## Case 2 Harsher penalties:

Observations: Lidar scan (1080).

Actions: accelerate, break, cruise.

Rewards: -1 for every step. -1000 for not completing. -500 for crash. 600 – time taken for completion of a lap

Terminated: when 2 laps are completed

Truncated: when car crashes.

Notes about the implementation: learning was done during the run itself.

NN: self.fc1 = nn.Linear(1080, 512)

self.fc2 = nn.Linear(256, 128)

self.fc3 = nn.Linear(128, 64)

self.fc4 = nn.Linear(64, 3)

Notes for modification: Found out that the model was breaking, but the breaking logic is not efficient enough. Ie the car wont slow down fast.   
2 possible modifications:   
a. actions: discretized versions of actual output speed  
b. actions: race, accelerate, cruise, decelerate, brake  
We chose a for further testing.

## Case 3 New actions:

Actions: discretized versions of actual output speed ie. Max speed of 7 divided into 0.5 buckets. Therefore, car speed = action \* 0.5. Therefore, 35 actions

Rewards: -1 for every step. -1000 for not completing. -500 for crash. 600 – time taken for completion of a lap

Terminated: when 2 laps are completed

Truncated: when car crashes.

Notes about the implementation: learning was done during the run itself.

NN: self.fc1 = nn.Linear(1080, 512)

self.fc2 = nn.Linear(256, 128)

self.fc3 = nn.Linear(128, 64)

self.fc4 = nn.Linear(64, 3)

Notes for modification: Car would randomly stop probably because during faulty runs, it would crash and hence the model would think just surviving is the best course of action.   
Maybe the penalties are too harsh

## Case 4 Scaled down rewards:

Actions: discretized versions of actual output speed ie. Max speed of 7 divided into 0.5 buckets. Therefore, car speed = action \* 0.5. Therefore, 35 actions

Rewards: -1 for every step. -50 for not completing. -10 for crash. -60 for 3rd crash. 300 – 4\*(time taken for completion of a lap)

Terminated: when 2 laps are completed

Truncated: when car crashes 3 times.

Notes about the implementation: learning was done during the run itself. Car has 3 lifelines.

NN: self.fc1 = nn.Linear(1080, 512)

self.fc2 = nn.Linear(256, 128)

self.fc3 = nn.Linear(128, 64)

self.fc4 = nn.Linear(64, 3)

Notes for modification: Car is not differentiating between different turns. Perhaps the model is too simple.

## Case 5 New Neural Network:

Actions: discretized versions of actual output speed ie. Max speed of 7 divided into 0.5 buckets. Therefore, car speed = action \* 0.5. Therefore, 35 actions

Rewards: -1 for every step. -50 for not completing. -10 for crash. -60 for 3rd crash. 300 – 4\*(time taken for completion of a lap)

Terminated: when 2 laps are completed

Truncated: when car crashes 3 times.

Notes about the implementation: learning was done during the run itself. Car has 3 lifelines.

self.fc1 = nn.Linear(1080, 512)

self.fc2 = nn.Linear(512, 512)

self.fc3 = nn.Linear(512, 128)

self.fc4 = nn.Linear(128, 35)

Notes for modification: Car sometimes gets competitive times of 16sec per lap.   
but mid training would stop doing so.  
Car can handle higher resolution of actions. So actions should be divided by 0.1 instead of 0.5.

## Case 6 Reduced Learning rate:

Actions: discretized versions of actual output speed ie. Max speed of 7 divided into 0.5 buckets. Therefore, car speed = action \* 0.5. Therefore, 35 actions

Rewards: -1 for every step. -50 for not completing. -10 for crash. -60 for 3rd crash. 300 – 4\*(time taken for completion of a lap)

Terminated: when 2 laps are completed

Truncated: when car crashes 3 times.

Notes about the implementation: learning was done during the run itself. Car has 3 lifelines.

self.fc1 = nn.Linear(1080, 512)

self.fc2 = nn.Linear(512, 512)

self.fc3 = nn.Linear(512, 128)

self.fc4 = nn.Linear(128, 70)

Notes for modification: Problem of car staying stationary persists. Model should be able to perform better than 16 secs. Suggestions: improve reward function. Car would decide to go from 0 to 7 in 0.01 secs causing the car to flip. There is no way to detect a flipped car in the simulator thus we cannot penalize the model for flipping the car. There is no way to give this feedback to the model hence, we need to programmatically avoid this possibility. We could add a acceleration function that takes the car from initial speed to desired speed with an acceleration function that doesn’t cause the car to flip.

## Case 7 Reduced Learning rate:

### Milestone: Car is able to get 16 secs. We can start fine tuning

Actions: discretized versions of actual output speed ie. Max speed of 7 divided into 0.5 buckets. Therefore, car speed = action \* 0.5. Therefore, 35 actions

Rewards: (current speed) for every step. -50 for not completing. -10 for crash. -60 for 3rd crash. 300 – 4\*(time taken for completion of a lap)

Terminated: when 2 laps are completed

Truncated: when car crashes 3 times.

Notes about the implementation: learning was done during the run itself. Car has 3 lifelines. Implemented the PID code versions acceleration function. Current speed output of acceleration function given the desired (predicted) speed.

self.fc1 = nn.Linear(1080, 512)

self.fc2 = nn.Linear(512, 512)

self.fc3 = nn.Linear(512, 128)

self.fc4 = nn.Linear(128, 70)

Notes for modification: As it is a realistic simulator using ROS. Sometimes there are network glitches causing the car itself to drive into the wall. This is also due to the fact that our system is running both the simulation and the model. We can modify training implementation such that the training happens only after an episode has completed.  
  
 Currently the car only has 0 or positive speed so no matter how slow the car moves, if the cars trajectory straight headed for the wall, it will crash into it.

Another improvement to the problem statement could be that the algorithm can not only lead to competitive times, but also reduced damage to the car if it is able to drive at a safe speed. Therefore, we can give a reward as a function of speed for crashes.

## Case 8 Moved Training:

Actions: discretized versions of actual output speed ie. Max speed of 7 divided into 0.5 buckets. Therefore, car speed = action \* 0.5. Therefore, 35 actions

Rewards: (current speed) for every step. -50 for not completing. -5\* speed for crash. -10\*speed for 3rd crash. 300 – 4\*(time taken for completion of a lap)

Terminated: when 2 laps are completed

Truncated: when car crashes 3 times.

Notes about the implementation: learning was done during the run itself. Car has 3 lifelines. Implemented the PID code versions acceleration function. Current speed output of acceleration function given the desired (predicted) speed. Training happens just before a car reset is required.

self.fc1 = nn.Linear(1080, 512)

self.fc2 = nn.Linear(512, 512)

self.fc3 = nn.Linear(512, 128)

self.fc4 = nn.Linear(128, 70)

Notes for modification: The Car once again is not learning to slow down at a turn. Need harsher penalties

## Case 9 Improved NN and Rewards: (modifying NN to see how it affects)

Actions: discretized versions of actual output speed ie. Max speed of 7 divided into 0.5 buckets. Therefore, car speed = action \* 0.5. Therefore, 35 actions

Rewards: (current speed) for every step. -500 for not completing. -50\* speed for crash. -100\*speed for 3rd crash. 240 – 4\*(time taken for completion of a lap). 0.5\* speed at every timestep.

Terminated: when 2 laps are completed

Truncated: when car crashes 3 times.

Notes about the implementation: learning was done during the run itself. Car has 3 lifelines. Implemented the PID code versions acceleration function. Current speed output of acceleration function given the desired (predicted) speed. Training happens just before a car reset is required.

self.fc1 = nn.Linear(1080, 2048)

self.fc2 = nn.Linear(2048, 512)

self.fc3 = nn.Linear(512, 256)

self.fc4 = nn.Linear(256, 90)

Notes for modification: Given the stochastic nature of the env. The model is finding it hard to converge. Will have to implement early stop to prevent model from over training.

The model also may need to see more observations as done by the Atari implementation of Deep mind.

## Case 10 Implement CNN:

Actions: discretized versions of actual output speed ie. Max speed of 7 divided into 0.5 buckets. Therefore, car speed = action \* 0.5. Therefore, 35 actions

Rewards: (current speed) for every step. -500 for not completing. -50\* speed for crash. -100\*speed for 3rd crash. 240 – 4\*(time taken for completion of a lap). 0.5\* speed at every timestep

Terminated: when 2 laps are completed

Truncated: when car crashes 3 times.

Notes about the implementation: learning was done during the run itself. Car has 3 lifelines. Implemented the PID code versions acceleration function. Current speed output of acceleration function given the desired (predicted) speed. Training happens just before a car reset is required. We take 4 frames of the observation and stack them as the input to the neural network. We then use a 1D Convolution and pooling on this input. We also reduced complexity of the Dense neural network

self.cn = nn.Conv1d(4, 8, 8, 8)

# self.pool = nn.AvgPool1d(4,4)

self.fc1 = nn.Linear(1080, 256)

self.fc2 = nn.Linear(256, 128)

# self.fc3 = nn.Linear(126, 70)

self.fc4 = nn.Linear(128, 70)

Notes for modification: 0.5\*speed reward at every timestep is making the model over learn that speed is better than safety hence even though it crashes, after 2-3 learning cycles it resorts back to high speed even at turns.

On removing the reward, the model barely even reaches the goal in time even after many timesteps of training.  
Hence to get the best of both, we will first train the model to learn to speed, once it converges, we will remove the reward and continue training the model. Thus the model will learn to now drive safer. We will use early stopping to prevent the model for overlearning safety (become too slow).

## Case 11 Shape Rewards:

Actions: discretized versions of actual output speed ie. Max speed of 7 divided into 0.5 buckets. Therefore, car speed = action \* 0.5. Therefore, 35 actions

Rewards: (current speed) for every step. -500 for not completing. -50\* speed for crash. -100\*speed for 3rd crash. 240 – 4\*(time taken for completion of a lap). 0.5\* speed at every timestep at episodes <1500. 0 at every timestep for episodes 1500-2000(end of training).

Terminated: when 2 laps are completed

Truncated: when car crashes 3 times.

Notes about the implementation: learning was done during the run itself. Car has 3 lifelines. Implemented the PID code versions acceleration function. Current speed output of acceleration function given the desired (predicted) speed. Training happens just before a car reset is required. We take 4 frames of the observation and stack them as the input to the neural network. We then use a 1D Convolution and pooling on this input. We also reduced complexity of the Dense neural network

self.cn = nn.Conv1d(4, 8, 8, 8)

# self.pool = nn.AvgPool1d(4,4)

self.fc1 = nn.Linear(1080, 256)

self.fc2 = nn.Linear(256, 128)

# self.fc3 = nn.Linear(126, 70)

self.fc4 = nn.Linear(128, 70)

## DQN Modifications:

In standard DQN the training happens during an episode. However given our implementation architecture, the car needs real time instructions. Else the car may crash into the wall while the cpu/gpu is busy training. Thus we modify DQN to train only after the episode ends.

To incentivise racing, we give a reward based on its current speed. However, this seems to override negative rewards of crashing and thus the car always accelerates even when its supposed to go slow. For this, we implemented Curriculum training, where first we train the car to race, once it converges, we train the car to drive safely. We can then early stop to pick an instance of the model that drives safely but also completes laps in a competitive time.